EXHIBIT 2

WILLKIE FARR & GALLAGHER LLP
BENEDICT Y. HUR (SBN: 224018)
bhur@willkie.com
SIMONA AGNOLUCCI (SBN: 246943)
sagnolucci@willkie.com
EDUARDO E. SANTACANA (SBN: 281668)
esantacana@willkie.com
LORI ARAKAKI (SBN: 315119)
larakaki@willkie.com
One Front Street, 34th Floor
San Francisco, CA 94111
Telephone: (415) 858-7400
Facsimile: (415) 858-7599

Attorneys for Defendant
GOOGLE LLC

UNITED STATES DISTRICT COURT

NORTHERN DISTRICT OF CALIFORNIA

SAN FRANCISCO

ANIBAL RODRIGUEZ AND JULIE ANNA MUNIZ, individually and on behalf of all other similarly situated,

Plaintiff,

vs.

GOOGLE LLC, *et al.*,

Defendant.

Case No. 3:20-CV-04688 RS

DEFENDANT GOOGLE LLC'S FOURTH SUPPLEMENTAL RESPONSES AND OBJECTIONS TO PLAINTIFFS' INTERROGATORIES, SET ONE

Judge: Hon. Richard Seeborg
Courtroom: 3, 17th Floor
Action Filed: July 14, 2020

PROPOUNDING PARTY: PLAINTIFFS ANIBAL RODRIGUEZ AND JULIEANNA MUNIZ

RESPONDING PARTY: DEFENDANT GOOGLE LLC

SET NO.: ONE

on that Google data collection, (c) what impact if any an app's disabling of analytics data collection has on that Google data collection, and (d) what impact if any an app's decision to use or not use any Google services apart from Firebase SDK has on that Google data collection.

RESPONSE TO INTERROGATORY NO. 1:

Google objects to Interrogatory No. 1 as vague and ambiguous as to the undefined term "Web & App Activity." For purposes of this response, Google construes Web & App Activity to mean the account-level setting called Web & App Activity. Google further objects to this Interrogatory as vague and ambiguous with respect to the phrases "Google's data collection," "impact" and "Firebase SDK." Google further objects that the definition of "Class Period" is vague and ambiguous, as the Interrogatory defines the term to mean "the class period in this case, as defined in the operative complaint," when the "operative complaint" has changed between when the Interrogatories were served and when these responses were provided, and the definition of "Class Period" differs between the original and amended complaints. Google further objects that the term "Class Period" is vague and ambiguous because it fails to provide a concrete range of time. Google further objects to this Interrogatory to the extent that it seeks information protected by the attorney-client privilege and/or the attorney work product doctrine. Google further objects to this Interrogatory as unduly burdensome, overbroad, and disproportionate to the needs of the Action because this Interrogatory seeks information outside of the Class Period, which has little to no bearing on Plaintiffs' claims.

Subject to and without waiving the foregoing objections, Google responds as follows: The type of information that can be collected through the Google Analytics for Firebase default implementation, if authorized by an entity that uses Google Analytics for Firebase, includes: (1) number of users and sessions, (2) first launches, (3) in-app purchases, (4) session duration, (5) screen views, (6) app updates, (7) operating system updates and (8) uninstalls. If an entity chooses to use Google Analytics for Firebase, it authorizes that the following parameters are collected by default with every event that is collected either by default or manually, as instructed by the app developer: (1) screen information, and (2) session information. Further, if an entity chooses to use Google Analytics for Firebase, it authorizes the following automatically-collected device-related dimensions: (1) operating system, (2) device model, (3) language, (4) app store, (5) app version, and (6) first launch time. Web & App activity is an account-level setting that functions independently from Google Analytics for Firebase. Accordingly, a user turning off Web & App Activity does not prevent apps from collecting data via Google Analytics for Firebase, which is a separate product that apps may choose to utilize to collect and analyze their own users' data.

SUPPLEMENTAL RESPONSE TO INTERROGATORY NO. 1:

Subject to and without waiving the foregoing objections, Google responds further as follows: Since its launch, Google Analytics for Firebase has collected the following types of information by default, if Google Analytics for Firebase was enabled by an app developer using the Firebase SDK: (1) number of users and sessions, (2) first launches, (3) in-app purchases, (4) session duration, (5) app updates, (6) operating system updates and (7) uninstalls. As of October 24, 2016, Google Analytics for Firebase has also collected screen views. If an entity chooses to use Google Analytics for Firebase, it authorizes that the following parameters are collected by default with every event that is collected either by default or manually, as instructed by the app developer: (1) screen information, and (2) session information. Google Analytics for Firebase has collected these parameters by default since its launch. Further, if an entity chooses to use Google Analytics for Firebase, it authorizes the following automatically-collected device-related dimensions: (1) operating system, (2) device model, (3) language, (4) app store, (5) app version, and (6) first launch time. Google Analytics for Firebase has automatically collected these device-related dimensions since its launch.

Web & App activity is an account-level setting that functions independently from Google Analytics for Firebase. Accordingly, a user turning off Web & App Activity does not prevent apps from collecting data via Google Analytics for Firebase, which is a separate product that apps may choose to utilize to collect and analyze their own users' data. The data logged by Google Analytics for Firebase in its default implementation are not collected if a developer has not enabled Google Analytics for Firebase. There are other products and features that are part of Firebase SDK that may collect overlapping pieces of data, such as device model, app updates, and other types of data, but all such functionality is publicly documented and intentionally implemented by app developers. Each such product and feature, none of which are implicated by the First Amended Complaint, separately obtain consent and publicly disclose their functionality.

Finally, footnote 13 of the First Amended Complaint cites https://firebase.google.com/docs/app-indexing/android/log-actions, which is a webpage dealing with a separate Firebase SDK product called "App Indexing." As the cited webpage notes, App Indexing cannot collect event data unless certain conditions are met, including that a user has turned their Web & App Activity Control to "on."

SECOND SUPPLEMENTAL RESPONSE TO INTERROGATORY NO. 1:

Subject to and without waiving the foregoing objections, Google responds further as follows:

Google carefully processes information it receives from app developers via Google Analytics for Firebase ("GA for Firebase"). As explained herein, entities that choose to use GA for Firebase must incorporate it into their apps, enable its function, and thereby cause it to begin logging event data that is subsequently uploaded to Google servers for analysis and reporting. Google takes measures to process event data sent to Google via GA for Firebase according to the type of consent each user gives. For example, Google runs consent checks on the data during several processing steps, and it only associates event data with a specific user when the user has consented to that, including by having the WAA control turned to "on." When consent is lacking, Google takes several sophisticated technological steps to ensure that event data cannot be associated with a specific user, including by imposing several technological barriers to what is known as unauthorized "joining" of logs together to de-anonymize users. Unauthorized joining is forbidden, and Google also takes several steps to ensure access to the logs in question is restricted. Google does not join user event data collected via GA for Firebase to re-identify anonymized data. Google's customers are likewise never given access to data that could be used to de-anonymize users.

Plaintiffs' First Amended Complaint cites and quotes public documentation about GA for Firebase to demonstrate the types of user app interaction data, or "event data" that Plaintiffs allege

Google is improperly collecting. In order for the data collection described in each of those pieces of documentation to occur, an app developer must enable GA for Firebase. If app developers don't enable GA for Firebase, the data that would otherwise be sent to Google servers by GA for Firebase isn't sent to Google servers. Google doesn't implement in Firebase SDK any "shadow" copy of GA for Firebase that operates regardless of whether a developer has enabled GA for Firebase, nor does Google has any "secret scripts" that ignore the consent settings of users in order to connect event data to specific users' profiles. GA for Firebase will not store event data connected to a specific user's profile unless the app has permitted it and a user has opted into certain features, namely: the supplemental checkbox under the WAA control (sWAA), which can only be turned on if WAA is also turned on; Google Ads Personalization (GAP); and the supplemental checkbox under the GAP control (NAC). Finally, based on a reasonable investigation, Google has been unable to identify any impact an app's decision to use or not use a Google product or service other than Firebase SDK would have on the functioning of GA for Firebase as described below.

With regard to the other products and features that are part of Firebase SDK that may collect overlapping pieces of data, such as device model, app updates, and other types of data, such functionality is publicly documented including on Google's public help center pages at GOOG-RDGZ-00013288 - GOOG-RDGZ-00013449, which can also be viewed at firebase.google.com and support.google.com.

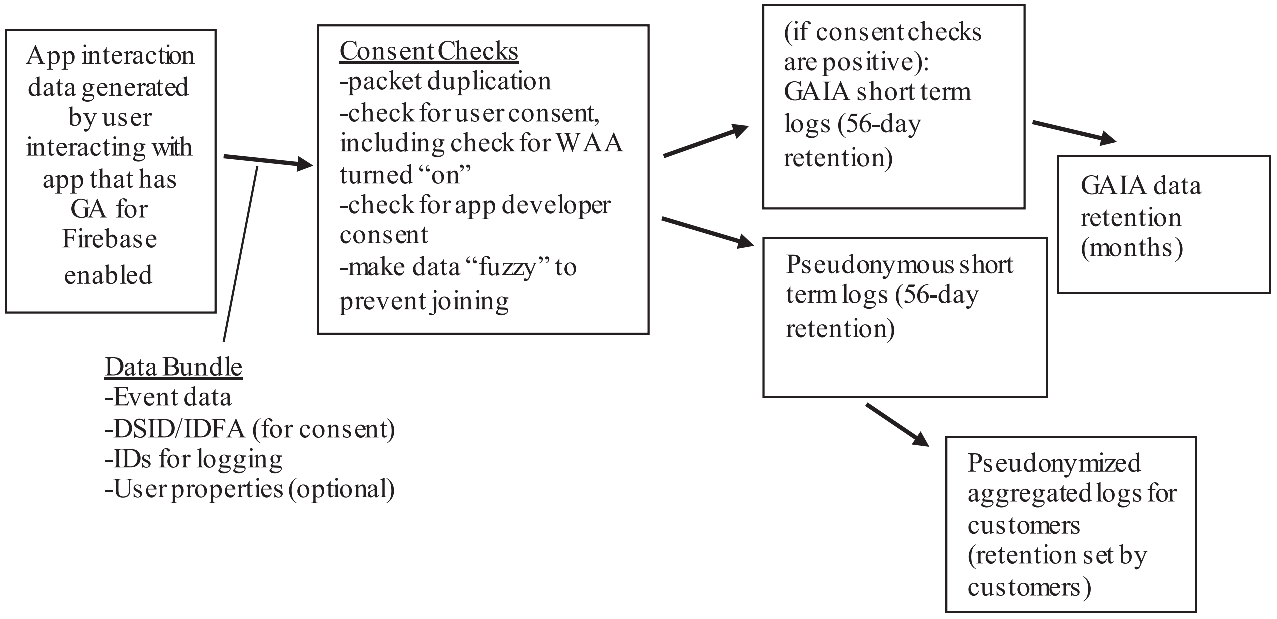
THIRD SUPPLEMENTAL RESPONSE TO INTERROGATORY NO. 1:

Subject to and without waiving the foregoing objections, Google responds further as follows:

Google carefully processes information it receives from app developers via Google Analytics for Firebase ("GA for Firebase"). As explained herein, entities that choose to use GA for Firebase must incorporate it into their apps, enable its function, and thereby cause it to begin logging event data that is subsequently uploaded to Google servers for analysis and reporting. Google takes measures to process event data sent to Google via GA for Firebase according to the type of consent each user gives. For example, Google runs consent checks on the data during several processing steps, and it only associates event data with a specific user when the user has consented to that, including by having the WAA control turned to "on." When consent is lacking, Google takes several sophisticated technological steps to ensure that event data cannot be associated with a specific user, including by imposing several technological barriers to what is known as unauthorized "joining" of logs together to de-anonymize users. Unauthorized joining is forbidden, and Google also takes several steps to ensure access to the logs in question is restricted. Google does not join user event data collected via GA for Firebase to re-identify anonymized data. Google's customers are likewise never given access to data that could be used to de-anonymize users.

Plaintiffs' First Amended Complaint cites and quotes public documentation about GA for Firebase to demonstrate the types of user app interaction data, or "event data" that Plaintiffs allege Google is improperly collecting. In order for the data collection described in each of those pieces of documentation to occur, an app developer must enable GA for Firebase. If app developers don't enable GA for Firebase, the data that would otherwise be sent to Google servers by GA for Firebase isn't sent to Google servers. Google doesn't implement in Firebase SDK any "shadow" copy of GA for Firebase that operates regardless of whether a developer has enabled GA for Firebase, nor does Google has any "secret scripts" that ignore the consent settings of users in order to connect event data to specific users' profiles. GA for Firebase will not store event data connected to a specific user's profile unless the app has permitted it and a user has opted into certain features, namely: the supplemental checkbox under the WAA control (sWAA), which can only be turned on if WAA is also turned on; Google Ads Personalization (GAP); and the supplemental checkbox under the GAP control (NAC). Finally, based on a reasonable investigation, Google has been unable to identify any impact an app's decision to use or not use a Google product or service other than Firebase SDK would have on the functioning of GA for Firebase as described below.

At a high level, the data flow of event data associated with GA for Firebase can be thought of conceptually as follows:


App interaction data generated by user interacting with app that has GA for Firebase enabled
Data Bundle
-Event data
-DSID/IDFA (for consent)
-IDs for logging
-User properties (optional)
Consent Checks
-packet duplication
-check for user consent, including check for WAA turned "on"
-check for app developer consent
-make data "fuzzy" to prevent joining
(if consent checks are positive): GAIA short term logs (56-day retention)
GAIA data retention (months)
Pseudonymous short term logs (56-day retention)
Pseudonymized aggregated logs for customers (retention set by customers)

Each of these steps is discussed in further detail below.

1. Data Logging

When app developers enable GA for Firebase, code authored by Google and activated by developers logs certain user interaction events automatically, such as the first opening of an app or when a user clicks on a certain part of the app. *See generally* https://support.google.com/firebase/answer/9234069?hl=en.

GA for Firebase on Android also logs: **DSID** (if the developer permits it, *e.g.*, if Google Signals is enabled for the Google Analytics property), which makes it possible to check against a specific user's privacy settings to ensure consent is provided; and **adid**, or "Ad ID," which is used for advertising purposes under certain circumstances as described more fully below.[1] On iOS, GA for Firebase logs **IDFA**, or "Identifier for Advertiser," which is the rough equivalent of the Ad ID on Android.

[1] *See* https://www.google.com/url?q=https://firebase.google.com/docs/analytics/configure-data-collection?platform%3Dandroid&sa=D&source=editors&ust=1623176595924000&usg=AOvVaw2T8Y_BUbEk1Od-0SpMEX0d.

GA for Firebase also logs the device's ads personalization opt-out setting: "opt-out of ads personalization, or OOOAP for Android and "limit ad tracking," or LAT for iOS.

GA for Firebase also logs a unique **app_instance_id**, which identifies the app session of the particular app running on a particular device. app_instance_id is not persistent; it can be refreshed when users take certain actions such as re-installing apps, certain updates to apps, and other similar actions.

Finally, for ads interactions, GA for Firebase also logs **aeid**, or "Ad Event ID," which is a unique identifier for the specific ad interaction logged in order to enable integration between GA for Firebase and AdMob if the customer has linked their AdMob app to Google Analytics.

This logging occurs at the same time as the interactions that trigger the event. For example, when a user clicks a specific button that the app developer has chosen to track using GA for Firebase, that button click is logged as it occurs. It is typically not uploaded to Google servers until later.

2. Data Upload

Apps with GA for Firebase enabled log and send information to Google via GA for Firebase code in a packet sometimes referred to as a "HitBundle." These packets contain (1) event information including app_instance_id, (2) DSID/IDFA for consent checks, and (3) non-personally identifiable information (PII) for user properties (if the app developer chooses to include such data). A HitBundle can contain multiple events grouped together. Event information includes different types of user-interaction data, as discussed above. The predefined and recommended events and user properties are publicly defined by Google. User properties are slowly changing data that describe the device or user of the device that an app developer may send, but it cannot send PII.

All of the HitBundle data is sent to Google in a single transmission. In other words, the app-interaction event data and user properties are sent to Google in the same packet as the DSID/IDFA, which is the information that allows Google to run consent checks.

For Android apps with Google Play Services enabled, GA for Firebase data is collected from all apps into a central file called app_measurement.db, which is periodically uploaded to

Google's servers. Google does this because it saves battery for users, whose devices would otherwise be initiating more uploads every day. On iOS devices, this is not possible, so each GA-for-Firebase-enabled app periodically transmits the data to Google's servers individually.

In all cases, data is first logged by GA for Firebase, recorded on the user's device in the appropriate file, and then subsequently uploaded to Google servers.

3. Consent Checks and Technical Barriers to Joining

HitBundles are received by Google at a "collection endpoint," where the data is stored in short-term memory. Before any data is ever written to disk, the server completes several important steps designed to protect user privacy. The first step is to check if the customer has enabled Google Signals. Everything described below regarding data duplication and consent checks is gated on whether the customer has enabled Google Signals. If the customer hasn't enabled Google Signals, the HitBundle is treated purely pseudonymously.

A preliminary step before the consent check occurs is data duplication. A single copy of the data packet received from a user's mobile device is made. This is done to facilitate the eventual data logging that respects user consent choices. One copy could become tied to a user's account if consent checks permit it; the other will become a pseudonymous log. During this time period, the copies are stored in short-term memory. The time between receipt and logging is typically less than five minutes.

Then, Google checks to see if a user is signed into their Google Account and whether they have opted into certain privacy controls, namely: the supplemental checkbox under the WAA control (sWAA), which can only be turned on if WAA is also turned on; Google Ads Personalization (GAP); and the supplemental checkbox under the GAP control (NAC). Each of WAA, sWAA, GAP, and NAC are account-level Web & App Activity and Ads Personalization controls. If sWAA, GAP, and NAC are all turned on, the result of the consent check is that the data can be tied to a user's Google account in what is known as "GAIA space." If any of these controls are off, then the data isn't joined to a user's Google account, and therefore not logged in "GAIA space."

The consent check actually happens in several steps, all in short-term memory, before any data is written to disk, *i.e.*, stored: (a) data is received at Google's servers and, if Google Signals is enabled, it is duplicated to facilitate logging; (b) Google uses a server to check whether a user is logged into a Google Account and whether consent has been given to join data to the account (the sWAA, GAP, and NAC settings discussed above); and (c) the data sets are cleaned and some data is regenerated to add additional obstacles to joining data.

To do this check, Google uses a DSID (IDFA on iOS). It tells Google who the user is by checking the DSID, which is an encrypted GAIA identifier. There will be no DSID in the HitBundle if the user is not signed into a Google Account. On iOS devices, the IDFA is associated with GAIA when users have logged into their Google Account and have not limited ad tracking. Google also checks to make sure the app developer consents to the tying of GA for Firebase data to users' accounts.[2]

Importantly, the Google server that receives the HitBundle, called [redacted], cannot decrypt the DSID. This is by design. Instead, [redacted] must send the encrypted DSID to a different Google server that performs the consent check and then returns to [redacted] either a "no consent" signal, or an encrypted GAIA ID. As a result, the hit bundles in short term memory on the [redacted] server cannot reflect a user's identity until a "yes consent" signal is received from a different server, which itself never receives the measurement data logged by GA for Firebase. All of this occurs in short term memory. The upshot of this is that the physical machine that receives the encrypted DSID from user devices isn't able to decrypt it, and the physical machine that decrypts the DSID doesn't receive the measurement data, it only gets the fields that can be used to check identity. This is all done to make it even less possible that a bad actor could infiltrate Google's system and perform an unauthorized "join."

[2] *See* https://www.google.com/url?q=https://support.google.com/analytics/answer/7532985?hl%3Den%23zippy%3D%252Cin-this-article&sa=D&source=editors&ust=1623176595926000&usg=AOvVaw1v59Uqg6gqH_XGgl4Z76xO.

If any aspect of the consent check fails, the user data is not stored in GAIA space, and the DSIS/IDFA is deleted.

After the consent check is completed and data is duplicated into two logs, Google removes data from each log to prevent "joining," which is a technical term that refers to re-identifying pseudonymized users by joining data from disparate parts of data held by Google together. Google takes a number of steps throughout its organization to prevent unauthorized joining of user data, and employees are, generally speaking, barred from doing it. Encryption is used to prevent Google personnel from identifying the data without authorization. Decryption keys are tightly controlled, and destroyed after a set period of time, making it impossible to re-join the data.

From the signed-in GAIA copy of data, Google removes all pseudonymous identifiers. From the signed-out pseudonymous log, Google removes all signed-in identifiers. The result is that the two logs don't overlap identifiers that could be used to join the logs together.

Google additionally regenerates data to prevent deterministic joining. A "deterministic" or "probabilistic" join would allow a complicated algorithm to use brute force and probability to guess which data belongs to which person. While the data is still in short term memory, Google creates "fuzziness," or slight errors, in data to prevent this. For example, Google adds random error into timestamps so they can't be matched together with timestamps elsewhere.

4. Differentiated Logging

Pseudonymous short-term logs have a 56-day retention period. They store the pseudonymized copy of the user interaction data—that is, they don't contain any identifying information, so it's not possible to tell to which user the event data belongs. They are used to create aggregated event data logs for customer use. GAIA short-term logs contain the same event data but keyed to a specific user.

To ensure that joining is practically impossible, Google takes several steps to scrub these logs of data that overlaps with data in other logs. Pseudonymous logs don't contain GAIA IDs. GAIA logs don't contain Device ID or app_instance_id. Both logs do contain aeid, or Ad Event ID, but in the pseudonymous log, aeid is encrypted with a 6-day retention key that is different from the encryption key used to encrypt Ad Event ID in GAIA logs. Further, in GAIA logs, aeid is "salted" as well, meaning random data is added to it to make it even harder to match it to the aeid stored in pseudonymous logs. As a practical matter, connecting these to each other is impossible.

5. Google's Encryption Technology

As further background, Google's encryption technology creates new encrypted keys every day. This enables the encryption keys to be retained for a fixed period (for example 60 days) of time for decryption purposes. Once the period of time passes, the keys are deleted, making decryption impossible. GA for Firebase also uses a different encryption key for the same user ID in GAIA space as compared to pseudonymous space (and in the case of aeid, in AdMob space). For that reason, it is prohibitively expensive to decrypt data from one log (e.g., pseudonymous) and join the data with another log.

For a set period of time, some IDs (device ID, app instance ID, and ad event ID) are stored and encrypted to allow Google to account for possible delays in data transmission and to fix any errors in data processing, including at customers' (i.e., app developers') request.

Device ID and app-instance ID are stored in encrypted form for 60 days. The reason the keys are stored for 60 days is because there are sometimes errors in the logs that require reprocessing of data. The 60-day window allows for customers to request reprocessing. It also allows Google to scan the logs to determine the amount of data affected by errors, which informs solutions to code issues. Google personnel are barred from using these keys without authorization, Google has monitors in place to ensure such unauthorized use doesn't happen, and access to the keys is tightly controlled. As a general matter, humans who work for Google would have no reason to gain access to these keys, and they don't. Computers do use them to reprocess data, as described here, but this happens without humans learning the keys, not unlike swiping a credit card at a grocery store.

Ad event IDs are stored in encrypted form for 6 days. These IDs are specific to a user's particular interaction with an ad. Ad event IDs allow Google to expand the Google Analytics pseudonymous data set with data from ADMob logs *if* the user has provided the proper permissions. Where a user interacts with a product that uses AdMob, that interaction is stored in AdMob logs and that data is joined to the Analytics data set using the ad-event ID.

The encryption keys are retained for 6 days to allow for processing time and errors. The processing happens daily and in some cases takes a few hours. If reprocessing of the data is necessary due to some error, the 6-day window allows Google time to do the reprocessing.

As discussed above, the ad event ID is the only unique ID common to both GAIA space and signed-out space that would allow for joining of data to a specific user. This means that for the 6-day window in which the ad event ID is stored with encryption, it is theoretically possible for someone with access to both ad event ID encryption keys to join data.

Google takes several steps to ensure this does not happen. As described above, Google generates new encryption keys each day and it uses a different key for the same user ad event ID in GAIA space as compared to pseudonymous space. Access to the encryption keys is tightly controlled and even the teams of engineers that create the code for GA for Firebase and related products/services do not have access to the keys. Google also protects the code that requires use of encryption keys. Once the 6-day period has elapsed, it is not possible for Google (or any other bad actor) to rejoin data stored in the pseudonymous logs with a user profile using the ad event ID.

Further, Google uses a restricted need-based access process and audit procedure. To begin, Google supports tiered access to the logs stored by GA for Firebase. None of the tiers includes personally identifiable information, but the default access level, for example, includes only event data, so Google employees cannot see any identifiers at all. Querying a log past the default state (i.e., geolocation logs or raw logs) requires a request ticket that allows Google to audit and track the request. Through that system, teams have to explain why they need a specific level of beyond-default access. Even if such access is granted, Google does not allow access forever. There are temporal restrictions: access is granted for a certain time frame after which teams will need to re-request access.

6. Use of User Data by Customers

User data is made available to external customers from whom the data originally was transmitted so they can conduct analytics and other marketing campaigns.

When data is reported to app developers, it is done so on an aggregated level. After pseudonymous logs are created, a subset of the data in them is then used to create the database for the back-end of GA for Firebase that feeds customers with aggregated reports. None of the ID data in that database overlaps with data in GAIA space, so no joining between them is possible. Both datasets do include event data, however, though timestamps are made "fuzzy" to prevent joining, as discussed above.

7. Use of Pseudonymous User Data by Google

Google uses user data collected via GA for Firebase across teams for product development, improvement, and diagnostics. As discussed above, if a user has provided consent, and if the AdMob and GA for Firebase Administrators at the customer have linked their AdMob account to their GA for Firebase app, Google can also join event data from GA for Firebase logs with AdMob log data in order to target advertising to users, all without personally identifying the user.

With regard to the other products and features that are part of Firebase SDK that may collect overlapping pieces of data, such as device model, app updates, and other types of data, such functionality is publicly documented including on Google's public help center pages at GOOG-RDGZ-00013288 - GOOG-RDGZ-00013449, which can also be viewed at firebase.google.com and support.google.com.

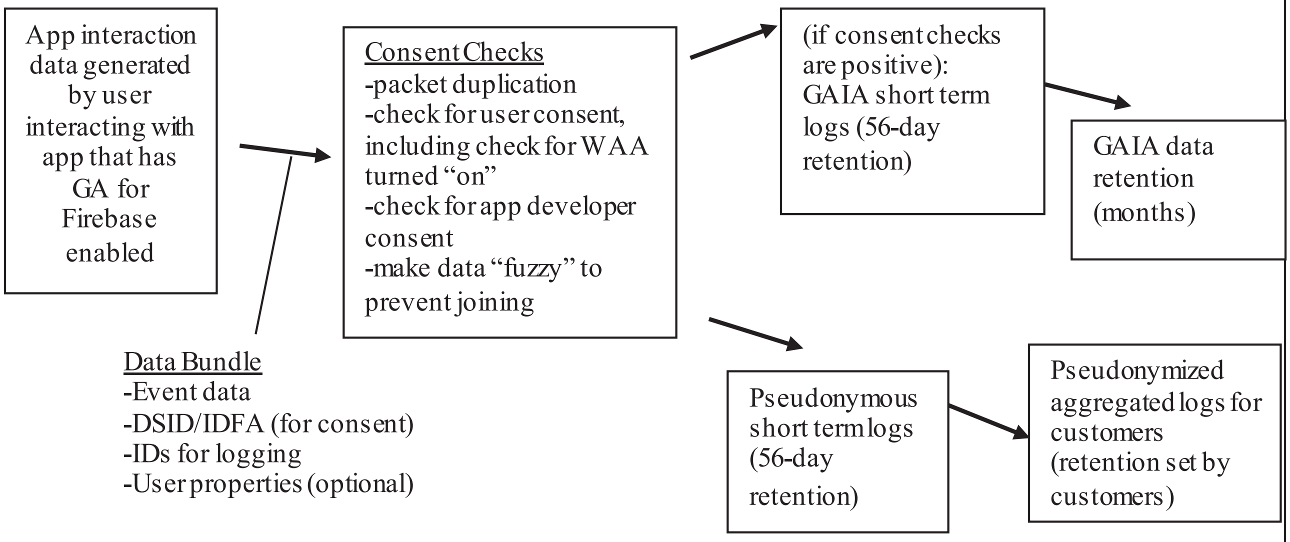
FOURTH SUPPLEMENTAL RESPONSE TO INTERROGATORY NO. 1:

Subject to and without waiving the foregoing, and without waiving any claim of privilege or protection of the work product doctrine, Google further responds as follows:

Google carefully processes information it receives from app developers via Google Analytics for Firebase ("GA for Firebase"). As explained herein, entities that choose to use GA for Firebase must incorporate it into their apps, enable its function, and thereby cause it to begin logging event data that is subsequently uploaded to Google servers for analysis and reporting. Google takes measures to process event data sent to Google via GA for Firebase according to the type of consent each user gives. For example, Google runs consent checks on the data during several processing steps, and it only associates event data with a specific user when the user has consented to that, including by having the WAA control turned to "on." When consent is lacking, Google takes several sophisticated technological steps to ensure that event data cannot be associated with a specific user, including by imposing several technological barriers to what is known as unauthorized "joining" of logs together to de-anonymize users. Unauthorized joining is forbidden, and Google also takes several steps to ensure access to the logs in question is restricted. Google does not join user event data collected via GA for Firebase to re-identify anonymized data. Google's customers are likewise never given access to data that could be used to de-anonymize users.

Plaintiffs' First Amended Complaint cites and quotes public documentation about GA for Firebase to demonstrate the types of user app interaction data, or "event data" that Plaintiffs allege Google is improperly collecting. In order for the data collection described in each of those pieces of documentation to occur, an app developer must enable GA for Firebase. If app developers don't enable GA for Firebase, the data that would otherwise be sent to Google servers by GA for Firebase isn't sent to Google servers. Google doesn't implement in Firebase SDK any "shadow" copy of GA for Firebase that operates regardless of whether a developer has enabled GA for Firebase, nor does Google has any "secret scripts" that ignore the consent settings of users in order to connect event data to specific users' profiles. GA for Firebase will not store event data connected to a specific user's profile unless the app has permitted it and a user has opted into certain features, namely: the supplemental checkbox under the WAA control (sWAA), which can only be turned on if WAA is also turned on; Google Ads Personalization (GAP); and the supplemental checkbox under the GAP control (NAC). Finally, based on a reasonable investigation, Google has been unable to identify any impact an app's decision to use or not use a Google product or service other than Firebase SDK would have on the functioning of GA for Firebase as described below.

At a high level, the data flow of event data associated with GA for Firebase can be thought of conceptually as follows:


App interaction data generated by user interacting with app that has GA for Firebase enabled
Data Bundle
-Event data
-DSID/IDFA (for consent)
-IDs for logging
-User properties (optional)
Consent Checks
-packet duplication
-check for user consent, including check for WAA turned "on"
-check for app developer consent
-make data "fuzzy" to prevent joining
(if consent checks are positive): GAIA short term logs (56-day retention)
GAIA data retention (months)
Pseudonymous short term logs (56-day retention)
Pseudonymized aggregated logs for customers (retention set by customers)

Each of these steps is discussed in further detail below.

1. Data Logging

When app developers enable GA for Firebase, code authored by Google and activated by developers logs certain user interaction events automatically, such as the first opening of an app or when a user clicks on a certain part of the app. *See generally* https://support.google.com/firebase/answer/9234069?hl=en.

GA for Firebase on Android also logs: **DSID** (if the developer permits it, *e.g.*, if Google Signals is enabled for the Google Analytics property), which makes it possible to check against a specific user's privacy settings to ensure consent is provided; and **adid**, or "Ad ID," which is used for advertising purposes under certain circumstances as described more fully below.[3] On iOS, GA for Firebase logs **IDFA**, or "Identifier for Advertiser," which is the rough equivalent of the Ad ID on Android.

GA for Firebase also logs the device's ads personalization opt-out setting: "opt-out of ads personalization, or OOOAP for Android and "limit ad tracking," or LAT for iOS.

[3] *See* https://www.google.com/url?q=https://firebase.google.com/docs/analytics/configure-data-collection?platform%3Dandroid&sa=D&source=editors&ust=1623176595924000&usg=AOvVaw2T8Y_BUbEk1Od-0SpMEX0d.

GA for Firebase customers can also choose to disable personalized advertising for the whole app, geographic regions, specific users, or specific events or user properties.[4]

GA for Firebase also logs a unique **app_instance_id**, which identifies the app session of the particular app running on a particular device. app_instance_id is not persistent; it can be refreshed when users take certain actions such as re-installing apps, certain updates to apps, and other similar actions. It is also refreshed automatically when customers reset their advertising identifier.

Finally, for ads interactions, GA for Firebase also logs **aeid**, or "Ad Event ID," which is a unique identifier for the specific ad interaction logged in order to enable integration between GA for Firebase and AdMob if the customer has linked their AdMob app to Google Analytics.

Event-logging occurs at the same time as the interactions that trigger the event. For example, when a user clicks a specific button that the app developer has chosen to track using GA for Firebase, that button click is logged as it occurs. It is typically not uploaded to Google servers until later.

2. Data Logging Technical Details

A comprehensive list of measurement data collected by Google Analytics for Firebase is here. Measurement data collected for Google Analytics for Firebase includes:

- IDFA/IDFV (on iOS); adid (on Android) as well as the LAT setting
- app_instance_id, which is a randomly generated identifier (UUID) on a device for a given app (plays a similar role to a first party cookie on the web)
- app_id, aka bundle_id (on iOS) and package_id (on Android)
- hash of developer cert (if applicable for the mobile platform)
- IP address (for geo lookup)
- information about the OS, OS version and device information
- First launch events, including timestamp of the first launch

[4] *See* https://support.google.com/analytics/answer/9626162?hl=en.

- In App Purchase events, including timestamp of the event, transaction ID, product ID, product name, price, currency code and quantity. IAP data will be collected automatically on iOS. The Google Analytics customer is required to link their Analytics property to Google Play to collect IAP data automatically on Android.
- collect referrer param string associated with the last campaign click that is attributable to a conversion

Google Analytics for Firebase also facilitates the logging of events related to Firebase Cloud Messaging (FCM). The FCM SDK logs the following events through Google Analytics for Firebase, when present and enabled:

- notification_receive : when the push notification was received by the app
- notification_foreground : when the push notification was received by the app while the app was in the foreground
- notification_open : when the user chose to open the notification
- notification_dismiss : when the user chose to dismiss the notification

While the FCM SDK does log debug messages corresponding to each of these events, those logs are entirely private to the app itself and can only be accessed by the app or by the developer when they have connected their debug device to their development host via USB. FCM does not log corresponding events to their own server. FCM uses the Google Analytics for Firebase standard means of logging events which results in these notification-related events being bundled and uploaded along with all other app events. Google Analytics for Firebase is the single owner and authority on the events enumerated above.

Each of these FCM events also logs corresponding event parameters to contextualize the push notification. These event parameters are:

- gcm_message_name: the name of the message in the FCM "Message Composer" UX

- gcm_message_time: the time at which the developer chose to broadcast push notifications. Alternatively, if the developer chose to use "device-local" time, message_time indicates a delivery time relative to each device's time zone.
- gcm_message_use_device_time : a flag which determines the interpretation of message_time.
- gcm_message_id: a unique ID for the message campaign
- gcm_sender_id: the sender ID corresponds to the developer's cloud project ID

These parameters correspond to fields in message campaigns launched by the GCM Product on the Web. None of these parameters are associated with users or devices.

These events are aggregated in tables in a manner consistent with all other events. Furthermore, app developers are able to send events and user properties to Google Analytics for Firebase. An event has a string name, and up to 25 event parameters (name / value pairs). It represents an event of interest happening inside the app on a particular device. Multiple events may be batched together into multiple HitBundle. In addition to events, a HitBundle may also carry user properties (name / value pairs), which are slowly changing data describing the device or the user of the device. App developers should not send PII in either event names, event parameters or user properties. Doing so is considered a violation of GMP TOS.

The app events data are joined with additional information on the server side, e.g., a device's screen resolution is derived based on a device model name. GA for Firebase collects app events data for mobile apps that are built with the Google Mobile Platform. The Firebase Analytics SDK also facilitates the logging of events related to Firebase Crashlytics.

The Firebase Crashlytics SDK will log the following events through Google Analytics, if present and enabled:

app_exception

In addition to logging this new event, Firebase Crashlytics captures the last N events that are logged on the device and -- upon an exception/crash -- reports these to their own BE.

The Google Analytics for Firebase SDK facilitates the setting of User Properties related to Firebase Cloud Messaging and Firebase Remote Config (aka Config).

The Firebase Cloud Messaging SDK will set the following User Property using the Google Analytics API, when present and enabled:

firebase_last_notification (working name)

The Firebase Remote Config SDK will set the following User Properties using the Google Analytics API:

firebase_experiment1_group (working_name)

firebase_experiment2_group (working_name)

The Google Analytics for Firebase SDK will collect the Instance ID (IID) for the purposes of Audience List exporting to Firebase. The SDK collects this ID from the user's device and includes it in the HitBundle.

On Android, the Google Analytics for Firebase SDK will collect the Android ID on devices that do not have Google Play Services installed. Devices without Google Play Services do not provide resettable_device_id (AdID).

The Google Analytics backend also aggregates per user cumulative metrics for Firebase A/B Testing (ABT). These metrics are used by ABT backends to determine the winning variation of multiple candidates. The access requirements of this data is same as all other aggregated Google Analytics data.

Google also supports view based campaign attribution (a.k.a. VTC) in addition to click from both adwords and 3rd party. If a user has not clicked on an ad previously, but has an ad view then Google will attribute the conversion to the ad viewed.

3. Data Upload

Apps with GA for Firebase enabled log and send information to Google via GA for Firebase code in a packet sometimes referred to as a "HitBundle." These packets contain (1) event information including app_instance_id, (2) DSID/IDFA for consent checks, and (3) non-personally identifiable information (PII) for user properties (if the app developer chooses to include such data). A HitBundle can contain multiple events grouped together. Event information includes different types of user-interaction data, as discussed above. The predefined and recommended events and user properties are publicly defined by Google. User properties are slowly changing data that describe the device or user of the device that an app developer may send, but it cannot send PII.

All of the HitBundle data is sent to Google in a single transmission. In other words, the app-interaction event data and user properties are sent to Google in the same packet as the DSID/IDFA, which is the information that allows Google to run consent checks.

For Android apps with Google Play Services enabled, GA for Firebase data is collected from all apps into a central file called app_measurement.db, which is periodically uploaded to Google's servers. Google does this because it saves battery and reduces network data costs for users, whose devices would otherwise be initiating more uploads every day. On iOS devices, this is not possible, so each GA-for-Firebase-enabled app periodically transmits the data to Google's servers individually.

In all cases, data is first logged by GA for Firebase, recorded on the user's device in the appropriate file, and then subsequently uploaded to Google servers.

4. Consent Checks and Technical Barriers to Joining

HitBundles are received by Google at a "collection endpoint," where the data is stored in short-term memory. Before any data is ever written to disk, the server completes several important steps designed to protect user privacy. The first step is to check if the customer has enabled Google Signals. Everything described below regarding data duplication and consent checks is gated on whether the customer has enabled Google Signals. If the customer hasn't enabled Google Signals, the HitBundle is treated purely pseudonymously.

A preliminary step before the consent check occurs is data duplication. A single copy of the data packet received from a user's mobile device is made. This is done to facilitate the eventual data logging that respects user consent choices. One copy could become tied to a user's account if consent checks permit it; the other will become a pseudonymous log. During this time period, the copies are stored in short-term memory. The time between receipt and logging is typically less than five minutes.

Then, Google checks to see if a user is signed into their Google Account and whether they have opted into certain privacy controls, namely: the supplemental checkbox under the WAA control (sWAA), which can only be turned on if WAA is also turned on; Google Ads Personalization (GAP); and the supplemental checkbox under the GAP control (NAC). Each of WAA, sWAA, GAP, and NAC are account-level Web & App Activity and Ads Personalization controls. If sWAA, GAP, and NAC are all turned on, the result of the consent check is that the data can be tied to a user's Google account in what is known as "GAIA space." If any of these controls are off, then the data isn't joined to a user's Google account, and therefore not logged in "GAIA space."

The consent check actually happens in several steps, all in short-term memory, before any data is written to disk, *i.e.*, stored: (a) data is received at Google's servers and, if Google Signals is enabled, it is duplicated to facilitate logging; (b) Google uses a server to check whether a user is logged into a Google Account and whether consent has been given to join data to the account (the sWAA, GAP, and NAC settings discussed above); and (c) the data sets are cleaned and some data is regenerated to add additional obstacles to joining data.

To do this check, Google uses a DSID (IDFA on iOS). It tells Google who the user is by checking the DSID, which is an encrypted GAIA identifier. There will be no DSID in the HitBundle if the user is not signed into a Google Account. On iOS devices, the IDFA is associated with GAIA when users have logged into their Google Account and have not limited ad tracking. Google also checks to make sure the app developer consents to the tying of GA for Firebase data to users' accounts.[5]

Importantly, the Google server that receives the HitBundle, called [redacted], cannot decrypt the DSID. This is by design. Instead, [redacted] must send the encrypted DSID to a different Google server that performs the consent check and then returns to [redacted] either a "no consent" signal, or an encrypted GAIA ID. As a result, the hit bundles in short term memory on the [redacted] server cannot reflect a user's identity until a "yes consent" signal is received from a different server, which itself

[5] *See* https://www.google.com/url?q=https://support.google.com/analytics/answer/7532985?hl%3Den%23zippy%3D%252Cin-this-article&sa=D&source=editors&ust=1623176595926000&usg=AOvVaw1v59Uqg6gqH_XGgl4Z76xO.

never receives the measurement data logged by GA for Firebase. All of this occurs in short term memory. The upshot of this is that the physical machine that receives the encrypted DSID from user devices isn't able to decrypt it, and the physical machine that decrypts the DSID doesn't receive the measurement data, it only gets the fields that can be used to check identity. This is all done to make it even less possible that a bad actor could infiltrate Google's system and perform an unauthorized "join."

If any aspect of the consent check fails, the user data is not stored in GAIA space, and the DSIS/IDFA is deleted.

After the consent check is completed and data is duplicated into two logs, Google removes data from each log to prevent "joining," which is a technical term that refers to re-identifying pseudonymized users by joining data from disparate parts of data held by Google together. Google takes a number of steps throughout its organization to prevent unauthorized joining of user data, and employees are, generally speaking, barred from doing it. Encryption is used to prevent Google personnel from identifying the data without authorization. Decryption keys are tightly controlled, and destroyed after a set period of time, making it impossible to re-join the data.

From the signed-in GAIA copy of data, Google removes all pseudonymous identifiers. From the signed-out pseudonymous log, Google removes all signed-in identifiers. The result is that the two logs don't overlap identifiers that could be used to join the logs together.

Google additionally regenerates data to prevent deterministic joining. A "deterministic" or "probabilistic" join would allow a complicated algorithm to use brute force and probability to guess which data belongs to which person. While the data is still in short term memory, Google creates "fuzziness," or slight errors, in data to prevent this. For example, Google adds random error into timestamps so they can't be matched together with timestamps elsewhere.

5. Differentiated Logging

Pseudonymous short-term logs have a 56-day retention period. They store the pseudonymized copy of the user interaction data—that is, they don't contain any identifying information, so it's not possible to tell to which user the event data belongs. They are used to create aggregated event data logs for customer use. GAIA short-term logs contain the same event data but keyed to a specific user.

To ensure that joining is practically impossible, Google takes several steps to scrub these logs of data that overlaps with data in other logs. Pseudonymous logs don't contain GAIA IDs. GAIA logs don't contain Device ID or app_instance_id. Both logs do contain aeid, or Ad Event ID, but in the pseudonymous log, aeid is encrypted with a 6-day retention key that is different from the encryption key used to encrypt Ad Event ID in GAIA logs. Further, in GAIA logs, aeid is "salted" as well, meaning random data is added to it to make it even harder to match it to the aeid stored in pseudonymous logs. As a practical matter, connecting these to each other is impossible.

6. Google's Encryption Technology

As further background, Google's encryption technology creates new encrypted keys every day. This enables the encryption keys to be retained for a fixed period (for example 60 days) of time for decryption purposes. Once the period of time passes, the keys are deleted, making decryption impossible. GA for Firebase also uses a different encryption key for the same user ID in GAIA space as compared to pseudonymous space (and in the case of aeid, in AdMob space). For that reason, it is prohibitively expensive to decrypt data from one log (e.g., pseudonymous) and join the data with another log.

For a set period of time, some IDs (device ID, app instance ID, and ad event ID) are stored and encrypted to allow Google to account for possible delays in data transmission and to fix any errors in data processing, including at customers' (i.e., app developers') request.

Device ID and app-instance ID are stored in encrypted form for 60 days. The reason the keys are stored for 60 days is because there are sometimes errors in the logs that require reprocessing of data. The 60-day window allows for customers to request reprocessing. It also allows Google to scan the logs to determine the amount of data affected by errors, which informs solutions to code issues. Google personnel are barred from using these keys without authorization, Google has monitors in place to ensure such unauthorized use doesn't happen, and access to the keys is tightly controlled. As a general matter, humans who work for Google would have no reason to gain access to these keys, and they don't. Computers do use them to reprocess data, as described here, but this happens without humans learning the keys, not unlike swiping a credit card at a grocery store.

Ad event IDs are stored in encrypted form for 6 days. These IDs are specific to a user's particular interaction with an ad. Ad event IDs allow Google to expand the Google Analytics pseudonymous data set with data from ADMob logs *if* the user has provided the proper permissions. Where a user interacts with a product that uses AdMob, that interaction is stored in AdMob logs and that data is joined to the Analytics data set using the ad-event ID.

The encryption keys are retained for 6 days to allow for processing time and errors. The processing happens daily and in some cases takes a few hours. If reprocessing of the data is necessary due to some error, the 6-day window allows Google time to do the reprocessing.

As discussed above, the ad event ID is the only unique ID common to both GAIA space and signed-out space that would allow for joining of data to a specific user. This means that for the 6-day window in which the ad event ID is stored with encryption, it is theoretically possible for someone with access to both ad event ID encryption keys to join data.

Google takes several steps to ensure this does not happen. As described above, Google generates new encryption keys each day and it uses a different key for the same user ad event ID in GAIA space as compared to pseudonymous space. Access to the encryption keys is tightly controlled and even the teams of engineers that create the code for GA for Firebase and related products/services do not have access to the keys. Google also protects the code that requires use of encryption keys. Once the 6-day period has elapsed, it is not possible for Google (or any other bad actor) to rejoin data stored in the pseudonymous logs with a user profile using the ad event ID.

Further, Google uses a restricted need-based access process and audit procedure. To begin, Google supports tiered access to the logs stored by GA for Firebase. None of the tiers includes personally identifiable information, but the default access level, for example, includes only event data, so Google employees cannot see any identifiers at all. Querying a log past the default state (i.e., geolocation logs or raw logs) requires a request ticket that allows Google to audit and track the request. Through that system, teams have to explain why they need a specific level of beyond-default access. Even if such access is granted, Google does not allow access forever. There are temporal restrictions: access is granted for a certain time frame after which teams will need to re-request access.

7. Use of User Data by Customers

User data is made available to external customers from whom the data originally was transmitted so they can conduct analytics and other marketing campaigns. The measurement data collected by Google Analytics for Firebase is used to provide Google customers with insights on app usage and user engagement. GA for Firebase allows sharing Analytics data with Google for improving Google products and services, enabling technical support, benchmarking, and sharing with Account Specialists. More on how shared data is used is here.

When data is reported to app developers, it is done so on an aggregated level. After pseudonymous logs are created, a subset of the data in them is then used to create the database for the back-end of GA for Firebase that feeds customers with reports. None of the ID data in that database overlaps with data in GAIA space, so no joining between them is possible. Both datasets do include event data, however, though timestamps are made "fuzzy" to prevent joining, as discussed above.

Raw event data may be exported to BigQuery for custom analysis. Google Data Studio can leverage raw Analytics event data through its BigQuery connector, which facilitates the generation of custom reports using Analytics events, parameters and user properties.

Users can adjust their data sharing settings in the Analytics Settings section in the Analytics UI.

Google only collects measurement data for apps that have the GA for Firebase SDK built in. Google's customers can obtain consent from end users to collect data. GA for Firebase customers can opt out of data collection according to the mechanisms discussed here.

8. Use of Pseudonymous User Data by Google

Subject to the data sharing setting "Share with Google to improve products and services," Google uses user data collected via GA for Firebase across teams for product development, improvement, and diagnostics. As discussed above, Google can also use pseudonymous event data from GA for Firebase logs to target advertising to users, all without personally identifying the user. Users can opt out of such ad targeting on their device by opting out of ad tracking on their Android or iOS device.

With regard to the other products and features that are part of Firebase SDK that may collect overlapping pieces of data, such as device model, app updates, and other types of data, such functionality is publicly documented including on Google's public help center pages at GOOG-RDGZ-00013288 - GOOG-RDGZ-00013449, which can also be viewed at firebase.google.com and support.google.com.

9. Data Storage and Access – Client-Side

Analytics data is stored in sqlite database on the device. The database is private and only the application has access to it. On Android Play devices the database is stored inside the GMS core services (aka Google Play Services) and the application doesn't have read access to the database. It can only use the client API to write to the database.

The local database stores the temporary raw events until they are uploaded to the Analytics collection endpoint. On successful upload the raw events are deleted from the local database. When the app fails to upload the raw events in 4 weeks data the data is discarded.

The local database also stores the user properties, the next sequential number for each event name, the timestamp of the last logged event and last set user property and the next sequential bundle index. Analytics data is uploaded in bundles that contain the current user properties, set of events that happen and app/device metadata.

Uninstalling the app clears all Analytics data. Any queued up data for uploading from the app might be uploaded after the app is uninstalled. Data queued up for uploading for more than 4 weeks will be discarded.

Uninstalling and reinstalling the app appears as a new fresh installation for Analytics. All existing data from the previous installation, including app_instance_id, is deleted. Calling resetAnalyticsData manually has the same effect.

Measurement Processing Pipeline

The Google Analytics for Firebase pipeline can be broken down into multiple components: event ingestion, event widening, event processing, aggregation and query serving. Event ingestion consumes the collection logs and creates hit bundles which are then output as either events that require special widening or ones that can be output for next stage i.e; event processing. Events that need widening go through a separate flow that takes care of adwords, doubleclick and other integration based event decoration. Finally post widening the events arrive at a stage that can be consumed by event processing. Event processing merges all these events and groups them in an optimized format, queries attribution service and also does read-modify-write of user state from user store [redacted]. Output of this stage is the raw baseview store stored in colossus files. Aggregation components consumes these baseview files to create specialized aggregate cubes. Finally query serving component uses [redacted] query to read from aggregates and serve responses to front end or API.

The infrastructure uses conduit as data processing and replication framework. [redacted] is used as execution framework for event processing stage of pipeline. Primary storage is in colossus files in capacitor format. For example; output of event ingestion job, baseview and aggregates. Google has used blobstore as backup storage for baseview and aggregates data.

10. Additional Data Protection Practices

User Data in GA for Firebase backends are kept in the following stores:

- [redacted] / [redacted] schema will enable [redacted] auditing and all manual access will be logged. Moreover jobs accessing this data needs to be BCID 3 compliant.
- Persistent CNS files (i.e. TTL longer than 10 days) with encryption keys kept in keystore.
    - There will be [redacted] logging on keystore access to these encryption keys.
    - The keystore config enforces BCID 3 compliance on access from production jobs.
    - Production jobs and individuals currently oncall will be granted access to these keys.
    - Developers will not otherwise have direct file access. Instead, they will be granted [redacted] read access to these files and the [redacted] access will be logged to [redacted].

For each GA for Firebase pipeline system component, the current oncalls (dev oncalls and SRE) have access to the production data storage. Additionally, break-glass access using access on demand can be obtained by off-call pipeline engineers to debug production issues. This access is granted for 20 hours. All storage is protected by [redacted] and encrypted.

11. Data Deletion

Analytics Measurement data follow the Analytics standard retention policy. Some of the key default storage retention periods in GA for Firebase include:

- Collection logs: 8 weeks
- Event ingestion output: 4 days
- Raw baseview: 60 days
- Aggregates: Kept indefinitely

INTERROGATORY NO. 2:

Please identify every app that includes or has included Google's Firebase SDK since the start of the Class Period, including for each app the time period during which that app used Google's Firebase SDK and Google therefore would have received data even when users had Web & App Activity turned off (or previously paused).

RESPONSE TO INTERROGATORY NO. 2:

Google objects to Interrogatory No. 2 as vague and ambiguous as to the undefined term "Web & App Activity." For purposes of this response, Google construes Web & App Activity to mean the account-level setting called Web & App Activity. Google further objects to this Interrogatory as vague and ambiguous with respect to the phrases "Google therefore would have received the data" and "Firebase SDK." Google further objects that the definition of "Class Period" is vague and ambiguous, as the Interrogatories define the term to mean "the class period in this case, as defined in the operative complaint," when the "operative complaint" has changed between when the Interrogatories were served and when these responses were provided, and the definition of "Class Period" differs between the original and amended complaints. Google further objects that the term "Class Period" is vague and ambiguous because it fails to provide a concrete range of time. Google further objects to this Interrogatory to the extent that it seeks information protected by the attorney-client privilege and/or the attorney work product doctrine. Google further objects to this Interrogatory because it is unduly burdensome, overbroad, and disproportionate to the needs of the Action, as it seeks a list of every app that includes or has